Organizing for openness : six models for developer involvement in hybrid OSS projects

Mäenpää, Hanna

Springer London
2018-08-16

Journal of Internet Services and Applications. 2018 Aug 16;9(1):17

http://hdl.handle.net/10138/238734

Mäenpää et al. Journal of Internet Services and Applications (2018) 9:17
https://doi.org/10.1186/s13174-018-0088-1

Journal of Internet Services and Applications

RESEARCH Open Access

Organizing for openness: six models for developer involvement in hybrid OSS projects

Hanna Mäenpää[1*], Simo Mäkinen[2], Terhi Kilamo[2], Tommi Mikkonen[1], Tomi Männistö[1] and Paavo Ritala[3]

Abstract

This article examines organization and governance of commercially influenced Open Source Software development communities by presenting a multiple-case study of six contemporary, hybrid OSS projects. The findings provide in-depth understanding on how to design the participatory nature of the software development process, while understanding the factors that influence the delicate balance of openness, motivations, and governance. The results lay ground for further research on how to organize and manage developer communities where needs of the stakeholders are competing, yet complementary.

Keywords: Open source, Hybrid open source, Governance, Community management, Software development process

1 Introduction

Using Open Source Software (OSS) removes many barriers regarding code reuse and modification. It offers companies many opportunities for speeding up new product development [1], by uncovering knowledge and work of highly qualified individuals that can be flexibly integrated into a company's value creation processes [2]. Collaborating with OSS communities can provide low cost means for early testing of quality and viability of products [3, 4]. This can leverage the capabilities of especially small and medium-sized companies [5, 6] – taken that they have sufficient resources and technological competencies for building effective and reciprocal collaborations [7].

Hybrid OSS communities can emerge either organically when companies become interested in existing projects or by design when companies initiate projects themselves by releasing software source code with an OSS-compliant license [5, 8]. When successful, this can invite stakeholders from different organizations into symbiotic relationships for creating a common software, while sharing also the same competitive market for software and services [9, 10]. This versatility introduces elements of competition in the collaborative software development effort [11] and emphasizes the importance of drawing the line between sharing innovations and maintaining private interests. While exchanging knowledge with a community is important for acquiring full benefits of the approach [12, 13]), the knowledge exchange can expose the company's assets and strategy or outsiders, including possible competitors [14].

From the viewpoint of a hybrid OSS development project, the varying aims of the stakeholders have a profound influence on how the future of the software is shaped. This effect is amplified by the stakeholders' capability to deploy resources, such as software developers into the development project [15]. These invisible power structures highlight the importance of fair goal alignment and incentivization of those actors who may not be able to advance their own aims as effectively as others, but who still are essential in ensuring sustainability of the development effort [12, 16].

As an OSS ecosystem grows, a central organization is often established to ensure stability and a common direction for the development community's work. This orchestrator can take an "open provider" role, where it facilitates the community's work with tools, infrastructure and social activities [4, 17, 18]. Another approach is that of a "closed sponsor" [5, 19], in which the orchestrator exercises centralized control over the community's platforms, processes and policies, controlling the

* Correspondence: hanna.maenpaa@helsinki.fi
[1]Department of Computer Science, University of Helsinki, Gustaf Hällströmin katu 2b, 00041 Helsinki, Finland
Full list of author information is available at the end of the article

ecosystem on various scope levels [19, 20]. This approach requires conscious decisions on how to structure the community to encourage - or limit participation [19] and how autonomy and centralized governance are balanced. If the community's members feel that their values are being compromised or that their opinions are not being heard, their motivation to contribute to the project can deteriorate [21, 22]. Here, distributing knowledge, autonomy and responsibility aptly can help to achieve a symbiotic state where the community sees the software ecosystem's members and especially the orchestrator as an active co-developer of the software, rather than a "parasitic" business owner that outsources its selected tasks to a crowd [2].

As for recent research, Alves et al. [9] call for concrete and actionable knowledge to help practitioners make strategic design decisions on the organization and governance of OSS ecosystems. This, as recommended by Linåker et al. [1] and Hussan et al. [23], can be studied at the scope level of the software development process by exploring how knowledge and access to development tasks are provided for the open developer community. We address these calls by illustrating how six orchestrators manage participation of external stakeholders in their hybrid OSS ecosystems by asking:

RQ1: How can decision-making roles and responsibilities be distributed?

RQ2: Which tasks of the development process can be accessible for members of the open developer community?

RQ3: What knowledge of the development process can be exposed for members of the open developer community?

While understanding that these issues are inseparable from their context, we provide a rich description of the case projects, their history and the current role of their orchestrator before answering the research questions. The work carries forward our previous work [24], where we examined three commercially oriented hybrid OSS developer communities in terms of their participation architecture. For this paper, we revisited the research questions that formed the core of our previous contribution, offering more precision to the results and adding three new case projects to the study. Our goal is to provide empirically grounded, illustrative analysis of the governance models in their genuine contexts. With this, we aim to discuss elements that can be used in designing and managing contemporary, hybrid OSS ecosystems. Next section describes developer community governance of hybrid software ecosystems based on literature. Section elaborates our research design and data collection strategy. Section 4 introduces the case communities, continued by findings in Section 5. Results are discussed and concluded in Sections 6 and 7.

2 Background and related work

Today, large OSS projects are typically arranged around a central organization that acts both as a guardian of the development community and as an orchestrator for its actions. While the nature of this organization (e.g. a foundation or a company) reflects the project's history, fundamentals of how a development community can form, act and evolve are grounded on the license of the software source code [5].

OSS licenses take different stands on users' rights to modify, re-use, and distribute the original software and its derivatives, along with what obligations stakeholders must adhere to when doing so [5]. The choice of a license has a profound effect on the project's nature and ability to attract new developers, and managers can use them to encourage contributions from the open development community [25]. Depending on the licensing scheme, the orchestrator can be allowed to package and re-sell the software [26]. This setting can structure the stakeholders to a single or multi-vendor ecosystem where individuals and businesses not only use and resell the software product, but also provide products and services for those who use it. When apt, the governance model of a developer community allows consolidating these various viewpoints into a coherent whole that supports the needs of the many and enables the ecosystem to grow in a sustainable manner [5, 27].

2.1 Governance considerations

Governance of OSS projects involves the means that are in place to steer the efforts of the autonomously working developer community [27]. Each community has its own governance model, which is an evolving configuration of coordination processes and practices that guide the community's action [28]. A model can emerge slowly as a collective learning process as the community finds its own tools and ways of working [4]. However, in the case of open sourcing a previously closed system, the governance model needs to be designed and implemented rapidly. Here, understanding which aspects and elements a model should consist of is crucial.

Governance styles vary between projects, and even within them, several layers of democratic, autocratic, oligarchic, federative and meritocratic principles and behaviors can coexist. These are implemented embedded in the various social interactions, written guidelines and standardized processes that operationalize the many authoritative elements present in the complex ecosystem [28]. A mature governance model explicitly documents

who the ecosystem consists of, how roles and responsibilities are distributed and ultimately: how contributions and their related decisions are made [20]. A fit model creates an efficient work environment where participants' motivation is fostered so that the community attracts and retains its developers [27]. From the orchestrator, this can involve taking an active stand on issues that are internal to the community, as well as its external relationships, such as who can enter the ecosystem and to which other projects the community's developers should contribute to [20] in order to ensure interoperability of the software or standardization of industry practices.

2.2 Accommodating participation

The first enabler of an open development project is the freely available software source code which in truly open communities is available for all stakeholders simultaneously in its complete form [29]. Next, the development project needs the ability to store knowledge about the software's defects and a means for receiving code contributions. While in the early forms of OSS projects, these two functions were based on email exchange, contemporary projects use tools that support and automate the contribution processes and related flows of knowledge [18]. These socio-technical systems constitute boundary objects of the community, providing personalized views on the current state of the community's work [30].

Using standardized tools facilitates the entrance of new developers [31] and allows the orchestrator to negotiate its relationship with the community from a more neutral ground as a part of the community's processes and practices are embedded in the features the software tools provide [28]. If the tool chain provides full transparency to the development process, external stakeholders can understand who the current members of the community are, what activities they are engaged in and at what state of the development process the current contributions are in [29]. The choice and configurations of these tools implement the governance model in terms of on what premises different roles, responsibilities and privileges are acquired and who can participate in distinct tasks [5]. These, together with a log of decisions made by the project's core developers, can open the development process to be freely observable and understandable for external contributors [29].

To increase community-drivenness, public decision support mechanisms can be used to choose who shall be accredited as new contributors and code maintainers [5]. At the same time, they can be used to resolve development priorities and contents of software releases. These decisions can be made via an open call, be restricted to a dominant control group, or kept to the central orchestrator [4, 5]. For limiting participation, the orchestrator can obfuscate access to development tasks or keep selected software assets proprietary, creating a "gate" that limits the open developer community's possibilities to understand and participate the development activities [21].

In mature communities, development standards and work practices may be enforced by automated testing of code contributions or by requiring contributors to perform complementary tasks, such as writing test cases and documentation [4, 32]. As for these complex nuances, a considerable amount of work can be required for making the community environment welcoming for newcomers and often personal mentoring is required for successful onboarding of new developers [33]. A special type of openness are the support mechanisms that are present for developers, such as documentation, tools and understandability of the participation and decision-making processes [29].

To summarize, while software licenses determine the many rights of users to copy, modify and distribute an Open Source software, they are only one component in the mix. A community's governance model defines how external stakeholders can view and influence the development project. Its openness or closeness is determined by access to tasks and transparency of knowledge to external contributors to the current state of the community and its activities. Managing a hybrid Open Source collaboration requires understanding and decisions on how "open" or "closed" a development project is for external contributors [29]. This degree of openness is multifaceted, and its interpretation varies according to the viewpoint. The main contribution of our paper is to illustrate how openness and closeness can manifest in the core of the development community - at the level of the software development process. To do this, we report empirical evidence from six hybrid OSS projects, highlighting how knowledge and access of development tasks can be used as leverages for openness.

3 Research design

Case studies can be used to investigate the industrial state of the practice in software engineering [34]. They can aim to extend the current body of knowledge about how contemporary phenomena manifest and evolve, yet also to create new theories that can be generalized and extended to new contexts with future research designs or practical applications [35]. Our research design is grounded on the work of De Noni et al. [4], who pinpointed as defining characteristics of community governance 1) the nature and role of the community's orchestrator and 2) the level of control it exposes to its community's operation and decision-making.

With this scope of investigation, we purposefully selected six Open Source Software development projects as our units of analysis. All projects have high commercial

involvement and a strong central orchestrator in place to steer the software development effort. However, each project represents a different flavor of organization and governance. Our sample includes four company-based and two foundation-driven Open Source development projects. We chose projects with both similar and different application areas, including both independent and interdependent projects in the mix. This strategy was chosen to collect versatile and illustrative examples of the build-up of contemporary hybrid Open Source communities and to highlight the differences of the way in which they operate.

3.1 Data collection and analysis

Embedded case study designs can bring phenomena to comparison in terms of the mutual characteristics they share [36]. Acknowledging this, we created an initial, theory-grounded framework for observing different aspects of 1) community-level [37], 2) project-level and 3) development process level governance and decision-making [5, 20]. In addition to knowledge and access to software development tasks, we wanted our analysis to cover the two paramount decisions that define the role of new developers: 4) how rights to make source code commits are gained and 5) how maintainership of specified areas of software source code can be achieved. This framework guided our inquiry from the freely available documentation and affordances that the projects' socio-technical systems provided for external contributors. An exploratory data collection strategy was chosen as explicitness of the governance model reflects its maturity [20], which we also wanted to discuss as a part of our findings. To increase validity of our findings, researchers conducted different stages of the data collection in pairs. When unsure, members of the development community were consulted for finding answers.

3.2 Research instrument

A community's governance model can be understood by examining the authoritative structures embedded in its coordination processes [28]. Respecting this, we crafted a research instrument to guide reporting our results. The instrument includes nine questions on where the case projects' governance models place their emphasis at the different stages of the software development process. The first five questions of our research instrument evaluate whether the open community's members were able to *access development* tasks by A) sending code contributions B) reviewing them and C) accepting the verified contributions to be integrated to the current development version of the software. In addition, we investigated whether external contributors were able to D) access the live development version of the software to e.g. verify defect reports and finally, whether members of the open developer community were able to E) impact priorities of the requirements. This, on its part, reflects how well the open community's members viewpoint was considered in short-term development decisions.

The remaining four questions illustrate how *knowledge* was exposed to external contributors about the status of the development process: whether developers could know F) which code contributions had recently been integrated to the development version of the software and G) what product planning and H) release definition decisions had been made. The last question assessed whether contributors were able to be on the pulse of the project by understanding the I) "live" development priorities of the project. Table 2 in the Findings section displays these questions and their answers in detail.

To provide a graphical representation of the results (Fig. 1), answers to these questions were coded based on a four-step numerical scale. We evaluated whether each matter was 0) kept proprietary to the central orchestrator, 1) revealed only to an exclusive group of the community's members, 2) open to individuals that had been accredited by either the community's members or the central orchestrator or 3) open to anyone interested. We claim that this approach presents a covering and sufficiently reliable viewpoint of an external observer on the practices in the case communities at the time of our observations in 2017.

4 Case projects

This section overviews the case projects in terms of their nature and history, setting the context for our research. Table 1 summarizes their main characteristics.

4.1 Eclipse

Eclipse is an integrated development environment (IDE), which is widely used by businesses and educational institutions as it supports a variety of programming languages and purposes of use. Eclipse is available with the EPL1 license, which places no restrictions on reuse or commercialization of the software. Therefore, the Eclipse community represents a typical multi-vendor ecosystem, where stakeholders base their core businesses on derivatives of the Eclipse software.

The software originates from IBM, which open sourced the project in 2001. Today, the Eclipse Foundation manages the project, hosting a partner ecosystem with a membership scheme of many levels based on the size of the stakeholder's business, its willingness to devote developer resources to the project and its desired position in the decision-making activities of the Eclipse development project. The foundation performs community building activities and actively hosts working groups, which e.g. develop industry standards for the software, such as Eclipse for the automotive industry and Eclipse for scientific use. The Eclipse ecosystem also encompasses a versatile set of

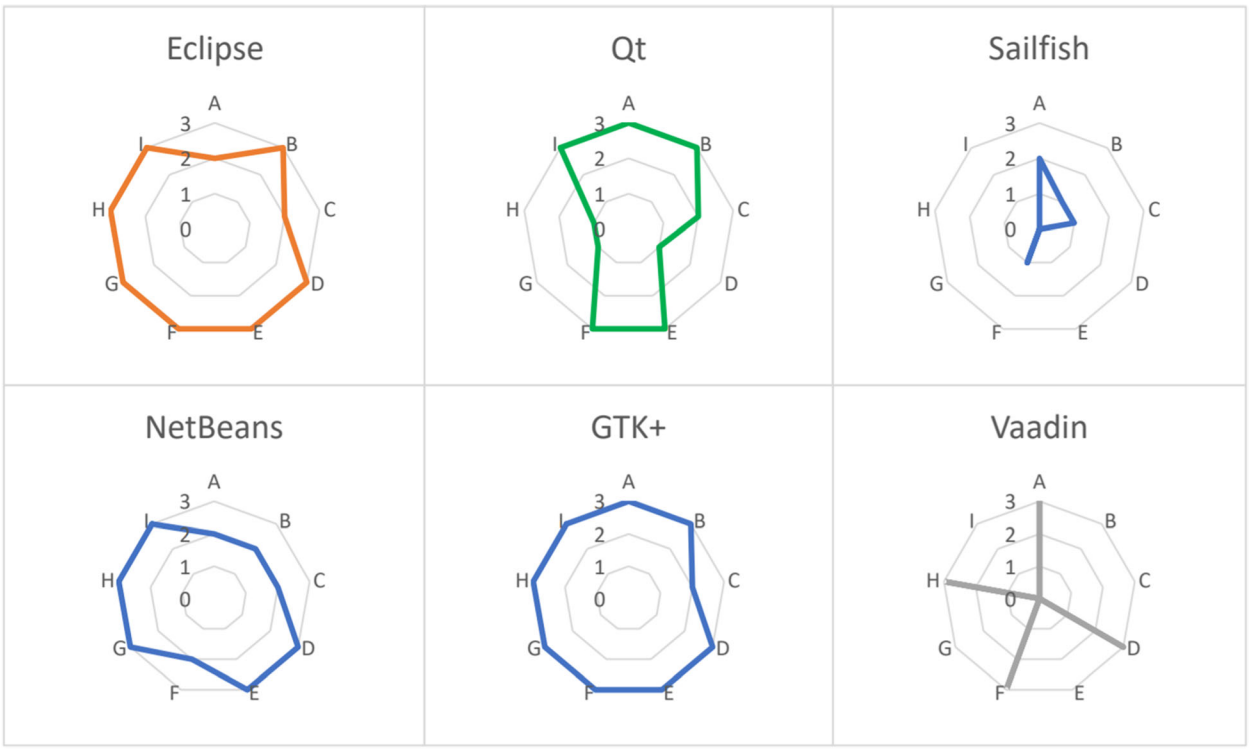

Fig. 1 Comparison of the case projects

independent software projects that build add-on components for the main platform. The source code of Eclipse is downloadable as a live, "debug" version. A new major version of the software is released once every year and several increments are made between. The development community is the primary source of release planning and definition and the foundation facilitates this through a standardized decision-making process.

4.2 NetBeans

Development of NetBeans IDE started as a student project in 1996. For the software's close relationship with

Table 1 Overview of the case projects

Software type	Eclipse 4.7 Integrated Development Environment	NetBeans 8.2 Integrated Development Environment	Qt 5 Application development framework	GTK+ 3 Application development framework	Vaadin 8 Application development framework	SailfishOS 2 Operating system
Language	Java	Java	C++	C	Java	C, C++, QML, Web technologies, shell scripts
Current contributors [44]	275	70	321	137	133	12
License	EPL1	GNU LGPL2.1	GNU GPL 3.0, LGPL 2.1 and 3.0	GNU LGPL 2.1	Apache 2.0	Proprietary, GPL, LGPL, BSD, and MIT.
Orchestrator	Eclipse Foundation	Oracle Ltd.	The Qt Company Ltd.	GNOME Foundation	Vaadin Ltd.	Jolla Ltd.
Position of the project	The foundation's primary interest.	Not primary business of the company.	Primary business of the company.	One of the GNOME Foundation's interconnected projects.	Primary business of the company.	Primary business of the company.
Orchestrator role	Active open provider, hub for member organizations.	Passive open provider.	Closed sponsor, deploys resources to development activities.	Open provider, organizes partnerships.	Closed sponsor, develops the software, supports the developer ecosystem.	Develops software, deploys resources to affiliated projects.

the Java programming language, Sun Microsystems acquired the NetBeans project from its original developers in 1999, immediately open-sourcing the software and continuing it as a community driven development project. In 2010, the ownership of both Java and the NetBeans project was transferred to Oracle, which let the NetBeans project operate independently, yet coordinating its two annual releases with those of the company's current Java platform.

The NetBeans software is licensed under GNU LGPL 2.1, which is intended to encourage development of both free and proprietary plugins to complement the IDE. Therefore, the NetBeans ecosystem invites numerous independently managed projects the work of which allows the IDE to support state of the art web development languages and frameworks that integrate it to the wider context of contemporary, industrial scale software development. After our data collection period was over, the NetBeans project was transferred from Oracle to the governance of the Apache Foundation. In this article, we focus on the community's state of matters before this major event in its history and leave the nature of this transformation to be a topic for future research.

4.3 Qt

Qt is a framework for building software applications for desktop, mobile, and embedded devices in domains such as industrial automation, medical devices, and in-vehicle entertainment systems. The project was started in 1991 by independent developers, who incorporated it in 1993 and released its software source code in 1995. After several re-definitions, the Qt software was licensed under GPL2 in 2000, which ensured the software's status as a common good. Following the commercial acquisitions by Nokia and Digia, the Qt project became hosted by the Qt Company, which today bases its primary business on a single vendor position by using a dual-licensing model. While the non-commercial version is Open Source, the commercial license allows making applications proprietary and to access complementary software components and personalized customer support. The Qt software is used widely in Linux-based environments, such as the KDE desktop environment and Sailfish OS, which is included in our study and described later in this section. For the KDE software's dependence on Qt, the KDE Foundation maintains the "Free Qt Foundation" to ensure that an open source version of the software will remain available. In addition to independent and commercial Qt application developers, the community's stakeholder ecosystem consists of consultancy companies and individual consultants who help their customers to build Qt applications, hardware manufacturers and different Open Source projects that build Qt related technologies. A major version of the Qt framework is released every 6 months, and several service releases are typically made between them.

4.4 GTK+

Similarly, to the Qt software, GTK+ is an application development toolkit that can be used to create graphical user interfaces for multiple platforms, including Windows, Linux, and iOS operating systems. It was originally developed in 1996 as a student project. Since then, it has been used mainly in products of the GNOME Foundation, such as the GNOME desktop environment and the GIMP graphics editor. The GTK+ development project has been orchestrated by the GNOME Foundation since 2000. Currently, the software is released under the GNU LGPL 2.1 license, which makes it possible to build GTK+ based applications for both noncommercial and commercial domains. The project's code maintainers are associated with commercial companies, such as Novell, Intel and Red Hat, and its ecosystem includes application developers and consultancies that help their customers in building their own applications with GTK +. Many OSS projects share inter-dependencies with the GTK+ framework. Therefore, developers may participate in the work of several related communities, and the bi-annual releases of GTK+ are defined largely by the work of the community's developers themselves.

4.5 Vaadin

Vaadin Ltd. produces an application development framework that is widely used for building interactive web applications for business use. The applications, written in Java, are developed using a charge-free version of the Vaadin framework, and they run on most operating systems and browsers. For accessing advanced features of the Vaadin framework, users can acquire a bundle of additional software components by purchasing a commercial license. The company hosts developer meetups and offers training for application developers.

The Vaadin software was initially developed as an add-on for an existing OSS product in 2002, and it was released as an independent software package in 2006. The development project has since been hosted by the Vaadin company that offers online training, consultancy and sub-contracting of application projects. The software is available under the Apache 2.0 license, which allows free modification and distribution of the software if the original copyright notice and disclaimers are preserved. However, distributing the Vaadin framework itself is not the primary aim of the ecosystem, which includes independent and commercial application developers, add-on developers and application development consultancies. The whole of the Vaadin software source code is publicly available and the community pre-release software version is available as a nightly build. The last

full version of Vaadin (8.0) was released in 2017. Minor releases are issued monthly.

4.6 Sailfish OS

Our last case is the Linux-based "Sailfish" operating system (OS) for mobile devices. This project was selected for its out-liar position: it is run by a commercial organization that displays a very closed governance model while employing a mixed licensing strategy in distributing its software – offering a contrast to the previous examples. The history of Sailfish OS originates from the Meego operating system software that was released from Nokia as open source in 2011. The current host of Sailfish OS is Jolla Ltd., a startup that sells both proof-of-concept mobile devices and distributor licenses for the OS. The technical architecture of Sailfish OS is layered, and parts of the software rely on the work of several Open Source communities, such as Qt and the independent, community-driven project Mer. The Mer software alone comprises of packages that are using various Open Source licenses, such as GPL, LGPL, BSD Licenses, and MIT License. The orchestrator's largest contribution to Sailfish OS are proprietary hardware-dependent Linux Kernel adaptations and closed source user interface libraries that define fundamentals of user experience of the operating system.

The Jolla company had chosen a gated source approach for Sailfish OS: it developed the hardware dependent kernel and user experience layers as an internal process and in isolation from the open developer community. The complete Sailfish OS software is available only in binary format, however, the company has experimented with different open innovation strategies for acquiring defect reports and testing for the proprietary components. The Sailfish OS project's immediate stakeholder ecosystem consists of users of Sailfish OS devices, the Mer project's software developers and hardware manufacturers that use Sailfish OS in their distributed devices.

With this set of six projects, we hope to provide a rich and versatile representation of possible governance configurations. The next section overviews the roles of the central orchestrator and describes the communities within the scope of the research questions that were presented in Section 1.

5 Findings

Guided by the framework described by Lindman et al. [38], we first lay out the support the orchestrator provided to its community and then proceed to describe the role of each orchestrator in their project's governance. Next, we describe how access to development tasks and transparency of knowledge were configured in each project. This is illustrated in Fig. 1, to which Table 2 reveals the research instrument's questions and coding of their answers.

5.1 Role of the orchestrator

All orchestrators sponsored platforms, tools and services to facilitate their open development project's work. Each ensured financial stability by bonding stakeholders to the project's ecosystem through partnering, customer relationships and memberships. Orchestrators offered legal and marketing support and helped with community-building activities. Most advocated actively to reach prospective new software users and developers. However, the level of the orchestrators' influence on their project's software development activities and related decision-making varied significantly.

The Eclipse Foundation maintained a complex, formal and hierarchical organization that was based on council and board memberships. A multi-leveled, fee-based membership scheme ensured the foundation's members special positions in the community's decision-making - in some cases also obliging them to deploy developer resources to

Table 2 The research instrument with questions and their coded answers

		Eclipse	Netbeans	Qt	Vaadin	GTK+	Sailfish
			~				
A	Who can contribute source code?	2	2	3	3	3	2[a]
B	Who can test code contributions?	3	2	3	0	3	1[a]
C	Who can accept code contributions?	2	2	2	0	2	1[a]
D	Who can access the complete, live development version?	3	3	1	3	3	0[a]
E	Who can impact work issue priorities?	3	3	3	0	3	0[a]
~							
F	Who knows integration status of code contributions?	3	2	3	3	3	1
G	Who can view the product roadmap?	3	3	1	0	3	0
H	Who knows release timing and content?	3	3	1	3	3	0
I	Who knows priorities of work issues?	3	3	3	0	3	0

0) Closed from outsiders, 1) Open only to an exclusive group, 2) Open to accredited individuals, 3) Open to any external observer

[a]Due to the distributed repository strategy, practices vary per software component

the project. The Eclipse Foundation applied standardized decision-making processes to accommodate viewpoints of both the corporate and independent developers, offering also a process for intellectual property rights clearance and mentoring to sustain architectural integrity and quality of the software. Daily work of the development community was organized in small and autonomous teams that each took care of a specific architecture module autonomously. Release definition of the software was notably community-driven, as sub-projects would suggest their own deliverables for each upcoming release.

Oracle stewarded the development of both the Java language and its affiliated development environment, NetBeans. However, it remained very passive towards the NetBeans project's governance. A management board with one representative of the orchestrator and two publicly elected community members was in place, yet not involved in the project's ongoing decision-making. The board existed to ensure that the project remained open and that conflicts between developers were resolved fairly. The project's support relied on a "Dream team" of committed individuals who were voluntarily in charge of the ecosystem-level activities. The development project acted autonomously, and community-level, project level, and technical decisions were at the sole responsibility of the open development community's members. Releases of both the Java language and NetBeans were synchronized, yet the NetBeans community's members had full autonomy in terms of release definition.

Similarly, the company-led Qt project showed a community-driven governance style, although in practice the orchestrator dominated the project by deploying its software engineers to the development community's work. A full software development pipeline was accessible and transparent for all stakeholders, yet the company's commercial customers and strategic affiliates held most of the technical leadership positions of the project. The orchestrator held to itself all decisions related to product planning and release definition. In addition to participating in the open development community's work in a transparent manner, the Qt Company offered several types of customer support, also building commissioned changes and new features to the software product through its internal software development process.

The Vaadin project invited both requirements, defect reports and code contributions from the open developer community. However, the development project was run by the company's employees and prioritization of work, release definition and long-term planning were at the sole responsibility of the orchestrator. Compared to The Qt Company, Vaadin presented a more transparent approach, constantly integrating small changes to their open repository. Still, the focus of the orchestrator was more on educating and building a community of Vaadin application developers than appropriating work from outsiders to the core framework. Both The Qt Company and Vaadin maintained technical and user documentation to support their developers' and application builders' work, whereas in the other projects, this was expected to be a community-driven effort.

The GTK+ project followed the GNOME Foundation's governance model. Membership of the foundation was free, accessible for all, and required for making contributions to the project. Membership granted a vote in the foundation's decision-making. The orchestrator hosted a portfolio of independent OSS projects, the work of which it packaged into the Gnome operating system software. Therefore, the foundation coordinated releases of its affiliated projects and aimed at unifying their development processes. An Advisory Board invited the foundation's corporate partners to enable communication and to strengthen the project's stakeholder ecosystem. A Board of Directors was in place to acquire sponsorships, to lay out the foundation's budget, and to manage staffing, legal issues, trademarks and public relations. The board was not directly involved in technical decisions, yet many of its elected members held technical leadership positions in the development community. As for commercial influence, the project's core maintainer team consisted of members of companies, such as RedHat, Novell and Google.

Jolla packaged the proprietary layers of the Sailfish OS with Open Source-based components and distributed the Sailfish operating system only as an executable binary, yet full source code was available with a written request from the company. Therefore, the orchestrator held release definition and product planning decisions to itself. In comparison to Vaadin and Qt, the Jolla company had a more outwards and networked approach as an orchestrator. Its employees worked in symbiosis with the open Mer OS project and directed defect reports and incoming code contributions to this project's issue tracking system. The orchestrator's engineers were dominant in steering the Mer community's work and they also participated actively in the work of related OSS communities, such as the Qt project, yet the emphasis of their work was mainly on the open Sailfish OS repositories and the Mer project.

5.2 Access to work tasks

Except for Sailfish OS, all projects disclosed the full software source code and offered a public contribution process for it. The main differences between the projects yielded from the way in which contributors could become actionable developers and how quality of the code contributions was ensured.

Due to both the Eclipse's technically fragmented architecture and the project's orientation towards small and autonomous teams, the project's requirements

management tool (Bugzilla) provided a complicated view to the project's development tasks. As each sub-project team managed their own source code repository independently, the sub-project's leader was held accountable for the quality of the work by the foundation's "release review"-process. For each release, the sub-project's contributions were integrated to the main development repository based on multi-stage release deadlines to which a sub-project committed by announcing its plans to the release management team. Before the final decision to integrate, the projects were subject to several formal reviews by the Eclipse Foundation's councils. Maintainership could be gained through self-nomination and a community vote by existing developers.

The contribution process of the NetBeans project was also open and community-driven, yet more centralized. While anybody could submit minor code increments to the project's mailing list and requirements management tool (Bugzilla), major development decisions were made by a core developer group who reviewed and integrated the contributions. The core maintainers had either created the architecture module or been accredited through a merit-based voting process by the existing maintainers. A write access to the development repository could be granted for developers after meaningful, small bug fixes. After showing their capability, a self-nomination and a consensus vote from existing developers was required. Maintainership could be gained by taking over an abandoned or starting a new sub-project or having it handed over by a current maintainer based on proven merit. Technical and community-level decisions were discussed publicly on the project's mailing list.

In the Qt project, submitting code contributions to the code review tool (Gerrit) was open for all and no limitations were in place for taking up either development-, testing- or code review tasks. Compliance of the contributions to the development standards was automatically tested, after which two humans were required to vote for accepting the code to the main development repository. Even though not strictly required, these typically were maintainers of the code area that the contribution dealt with - and most maintainers were associated with either The Qt Company or its affiliates KDAB Group GmbH and Intel. However, in principle this position was accessible for merited developers despite their organizational affiliation. After the code review stage, contributions were automatically integrated to the current development repository and the software was also delivered as nightly builds. Decision-support functions of the project's workflow management tools (Jira) allowed external developers to view requirements, verify existence of defects and to study their nature as an open and communicative process.

The Vaadin project welcomed everybody to contribute code and test the software, yet the company required a large amount of complementary contributions to accompany code increments sent by external developers. The full development version's repository was accessible, and the project had an openly available code review tool, from which also the status of each contribution was visible. However, in practice only the company's employees were accepting new contributions - in a communicative process with the associated developer - and maintaining the software source code. However, the company had recently started seeking periodical contributions through defect fixing campaigns [39], reaching out for a more interactive relationship with their community.

The GTK+ project welcomed contributions to the project's issue tracker (Bugzilla) and developer mailing list from all. Write access to the GTK+ project's development repository needed to be applied from the GNOME Foundation, and it was explicitly stated that the decision was not based solely on the contributor's previous merit. Also, here the core developers were making the final decisions on which code to integrate to the development version and the emphasis of the external contributors' work was more on reporting, testing and fixing defects than developing the core framework. Maintainership was based on a contributor's merit and discussed on the developers' mailing list.

While dominating the development project in both the proprietary and open components of the Sailfish OS and its affiliated Mer OS project, Jolla offered an online question and answer forum for contributors who provided user assistance, product planning feedback and detailed defect triage information. Even though the contribution process in the open components was accessible for any contributor, the distributed repository strategy encumbered entry of new contributors significantly. Aspiring contributors needed to first find the appropriate repository where their contributions should be submitted, which was an overly demanding task for newcomers. Maintainership was nominally open for all, yet in practice both Mer and the open repositories of Sailfish OS were maintained by the company's employees. In Fig. 1 and Table 2 we provide additional detail to the accessibility of development tasks. This view has been solicited by accessing publicly available documents and development tools, and therefore it is based on the external observer's view on openness of the community at the time of our observations during 2017.

5.3 The project's pulse and future

As explained above, the communities had versatile strategies for product planning, ranging from genuinely transparent and community-driven (NetBeans, GTK+, Eclipse) to collaborative (Sailfish OS) and even

possessive (Qt, Vaadin). Questions F-I of Table 2 provide additional detail to the transparency of the state of the development process, product- and release planning.

In the case of Eclipse, daily status of the work was efficiently known only at the sub-project level where developers interacted most frequently. The project's release definition was based on the sub-projects commitment to their own, announced goals. Here, the role of the orchestrator was to coordinate and communicate and to offer mentoring to support the sub-projects to achieve their targets.

In both the GTK+ and the NetBeans projects, having a closely-knit core developer group and an emphasis on mailing-list based communications created ambiguity of the daily status and priorities of the development project. However, roadmaps and release goals were disclosed publicly in a very specific manner.

At the other extreme, the Qt project provided a completely transparent end-to-end view to the development pipeline by disclosing the integration status of code contributions with its code review tool. However, the orchestrator communicated an overview if its long-term goals at a very general level. To compensate, the orchestrator published memos of the company's internal release team's online meetings, Here, due to the varying abilities of the developer ecosystem stakeholders to deploy developers to the project, invisible power structures had a profound influence on the development effort and transparency of its priorities.

In the case of Vaadin, the release- and product planning decisions were solely at the responsibility of the orchestrator. Even though online collaboration tools revealed timely information about the project's status, the project's priorities were known only by the company's engineering team. Vaadin visibly sought input for their next software release from the public by allowing them to vote from a pre-selected set of features and published both release themes and milestone-specific task lists in their Github issue manager. However, the company neither revealed the full contents of the releases, nor expressed commitment to any scope of these goals.

Even though Jolla was very cautious in disclosing the Sailfish OS release dates and product plans, the orchestrator held weekly online meetings to synchronize with their supportive projects' contributors. These meetings were public so that external stakeholders could understand the progress at a coarse level and some development decisions could be traced back to their source. The company published themes for the releases yet kept exact stage and plans of the development proprietary. Additional transparency was provided by issuing frequent "community" releases for the downloadable, bundled operating system.

6 Discussion

Hybrid Open Source projects accommodate work from stakeholders with versatile and often competing interests [11, 14]. Here, the importance of goal alignment and management of both the stakeholders, their expectations and contributions becomes highlighted. The different governance models described in our study reflect the history and mission of the central orchestrators [15, 19], at the same time enabling complex relationships and dynamics between the actors. In each case project, varying levels of trust and autonomy coexisted towards the open developer community, which was found to be the sum of several issues in the communities' governance model.

6.1 Inviting participation

In terms of distributing development tasks, decision-making roles and responsibilities (RQ1), the orchestrator's influence can span all aspects of the development community's operation. It can range from ecosystem level to community organization, project management and technical decisions. This can require active leadership through building communications strategies and designing operational support throughout the whole life cycle of the project. Here, the focus and scope of the orchestrator's control offers an interesting viewpoint to the case communities.

In both the Eclipse and Qt projects the orchestrators' earnings logic required them to ensure that some stakeholders were offered more influence on the development community's work than others. In Eclipse, this was done through membership-based decision-making mechanisms, whereas in the Qt project, the stakeholders' influence was directly ensured by offering customers both the orchestrator's internal software development unit's work and a possibility to gain a direct commit access to the development repository. In the third framework-based project Vaadin, the orchestrator's strategy was to service as wide and versatile application developer ecosystem as possible, which in their case emphasized the orchestrator's need to control quality and product development decisions over servicing the needs of individual stakeholders, as was the case in the Qt project.

We hypothesize that the GNOME Foundation aims at increasing interoperability of its portfolio projects by standardizing their development processes, platforms and schedules. Similar synchronization was in place in the NetBeans project, yet the only control point was found to be timing of the community's action. This could be due to the nature of the products, as the GTK+ software was used as a component, rather than a stand-alone software. Similarly, to the GNOME Foundation, the Jolla company composed its product from the work of several projects. However, as a relatively new project, the orchestrator was only opening its

development approach with the aim of securing opportunities for its future customers to participate in adapting the software. Therefore, its goal seemed to be to control its affiliated projects to gain a maximum of benefits from their work.

6.2 Managing contributions

The orchestrators had different means for controlling their development project and its related ecosystem. Examples of these are summarized in Table 3. As a general finding, offering access to a public repository, work issue trackers and code review tools did not guarantee an un-obscured view to the development project and its priorities. Even though e.g. release planning information was available, in most cases it was not detailed enough to give a fully transparent view to the project.

In terms of the different development tasks (RQ2), access to the full software source code formed the most powerful leverage of openness. At the same time, creating a gated approach could be done through proprietary licensing, distributing work to small and autonomous teams or developing parallel branches of the complete source code individually. These were found to be effective for limiting participation by obfuscating development tasks and their status from aspiring contributors.

Also, the choice between providing live access to the most recent software version was used for creating different "tiers" of stakeholders that view the development process from different perspectives. Varying knowledge about the current state and plans of the development work were used in a similar fashion: to either encourage or limit participation.

Table 3 A summary of findings

Form of control	Examples
Stakeholder control	Product pricing and membership fees. Strategic partnerships and special treatment of selected stakeholders. Offering advisory- and board memberships. Requirements for contributions, initiation rites. Disclosure of the ecosystem's members.
Process control	Nature and form of decision-making and software development processes. Milestones and decision points. Choice of supportive platforms and development tools. Full or partial disclosure of the development processes.
Assets control	Means for storing and processing requirements and other data. Documentation. Offering limited access to source code based on time-, scope level or requirements for stakeholders.
Leadership	Acquisition of new developers and software users. Campaigns for increasing contributions. Communicating product vision, release themes and timing.

6.3 Sharing the cost of openness

All in all, the openness does not come without a cost, as it often requires major inputs and investments from the ecosystem's orchestrator. How these costs are covered and how the orchestrator is incentivized to support the openness is an important question. It is worthwile to consider which part of the development process benefits the most from feedback - and who should act according to it, if any.

In the case of Eclipse, autonomy of the sub-projects came at the cost of deadline pressure and possible management overhead. In the case of Qt, community-drivenness required extensive tool support and resource deployment from strong, commercial organizations with high technological skills and in many cases a direct relationship with the company. For Vaadin, Jolla and The Qt Company, securing the orchestrator's strong position necessitated an active approach from its employees for the software development to sustain its focus. This, on its part, raised the costs of orchestration. In the more community driven projects GTK+ and Net Beans, this responsibility was distributed to the community's external stakeholders in a more equal manner. Here, the collaborative decision-making was ensured by the communities' consensus-driven accreditation process of new developers. Understanding these kinds of tradeoffs provides an interesting viewpoint to planning both value creation and delivery processes of the orchestrator.

6.4 Theoretical implications

Our results contribute to the interface of Open Source and Open Innovation literature by showing how software engineering processes can be built and managed to accommodate open innovation strategies [23], and what decision factors should be considered in doing so [1]. This can be contextualized to the community-type organizing of open innovation activities [16] and thus, contribute to the discussion of how open community-type of innovation activities can be organized in cases where a focal actor is orchestrating or facilitating the development community environment [40]. Second, our results imply that the tension between private-collective innovation incentives (see [41, 42]) can be addressed through considerations on how the different dimensions of governance support or restrict community-drivenness of the development organization. This tension in private-collective interests constitute an ongoing debate in research and development relationships and networks (see e.g. [43]), and managing them is decisive in open innovation communities and projects with diverse stakeholders.

Here, future research could examine how openness affects other design choices in the orchestrator's value creation processes, and to the way in which the ecosystem's partnering is organized. Further research could ask: as

openness grows, what are the other design choices that need to be modified in the platform to ensure innovativeness and capture of value from the ecosystem? As openness is seen as both necessary as well as risky strategy from the point of view of commercial actors [14], more understanding is needed on reaching the right balance between intellectual property protection, openness, and innovation potential of ecosystems [26].

6.5 Limitations

Governance models can evolve throughout time and in many cases, these changes are undocumented. Our data collection period took place in June and December 2017 and therefore the results represent the state of the practices in the case communities at those given points of time. Also, during the validation stage of our first findings, we recognized that the viewpoint of the observers can greatly influence their perception of the community's openness. What seems open from within the central organization, may not be so from the external contributor's point of view.

Our empirical approach does not allow generalization of the findings to larger contexts [34, 35], however we claim our study to be illustrative of the versatility of a possible governance configuration. Regarding the connection between the community's history and its governance model, we remark that a more focused and longitudinal research approach would be required for drawing conclusions on why certain models have emerged and exist. We leave this as an idea for future researchers, also calling for studies of transitions from company-led to foundation-driven governance. Of interest could also be to study how changes in the sociotechnical tools of a community can act as a change driver for its governance.

7 Conclusions

The work of Open Source Software development communities has already demonstrated its ability to disrupt software businesses by enabling faster product development cycles, cutting development costs and allowing new business concepts that in turn are creating new markets in the form of different service offerings and licensing models. At the same time, the OSS field suffers from the different goals and values of the involved stakeholders, which can range from almost purely ideological thrive to fundamentally commercial and often competing interests. This environment calls for reciprocal value creation models where stakeholders must have an opportunity to meet their own goals – simply to motivate them to play a role in the community.

In this paper, we contribute to this field in two ways. We investigate how hybrid OSS communities can be built to accommodate developer ecosystems and examine governance configurations of six longstanding software development organizations. We find that the role of the open development community and the principles according to which individuals can meaningfully participate in its activities are essential design factors of the hybrid OSS development model.

Managing this environment requires careful consideration on how much knowledge and influence should be released to the open development community and what the impact of this openness is. Also, we encourage considerations on how contributions can be managed by using the many different roles and requirements that can be defined by a community's governance model. These decisions also influence the scope of the community's autonomy and followingly the extent to which it can steer the future of the software product.

Organizations that use the hybrid OSS development model can act in different ways, leveraging either closed or open practices to support their own interests and revenue earning model. As a trend in our six cases, modernization of the socio-technical systems allowed a more open and distributed decision-making model. However, we emphasize that these conclusions should not be extended other contexts than presented in our study. Still, our observations can provide valuable insights for both researchers and practitioners that design and manage contemporary, open development communities that stem from similar settings as our case projects.

Abbreviation

OSS: Open Source Software

Funding

This work was funded partially by the "Innovative Requirements Engineering Methods, Algorithms and Tools" research project (OpenReq), which received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 732463. Researchers were granted full autonomy in designing and conducting the research.

Availability of data and materials

Please contact author for data requests.

Author's contributions

HM is responsible for preparing the manuscript, including the literature review and overall design and coordination of the study. SM has helped the first author to reliably gather data for the GTK+, Eclipse and NetBeans on for answering RQ1-RQ3. The research instrument was designed in collaboration with TK who is also responsible for gathering data for the Vaadin case and triangulating the results for Qt, Vaadin and Jolla cases with the first author. TMi, TMä and PR have helped by reviewing and commenting the paper and its various working versions. All authors read and approved the final manuscript.

Ethics approval and consent to participate

Not Applicable

Consent for publication

Not applicable

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]Department of Computer Science, University of Helsinki, Gustaf Hällströmin katu 2b, 00041 Helsinki, Finland. [2]Department of Pervasive Computing, Tampere University of Technology, Korkeakoulunkatu 1, FI-33720 Tampere, Finland. [3]School of Business and Management, Lappeenranta University of Technology, Skinnarilankatu, 34 53850 Lappeenranta, Finland.

Received: 11 October 2017 Accepted: 15 June 2018
Published online: 16 August 2018

References

1. Linåker J, Regnell B, Munir H. Requirements engineering in open innovation: a research agenda. In: proceedings of the 2015 international conference on software and system process. ICSSP. 2015;2015:208–12. https://doi.org/10.1145/2785592.2795370. ACM
2. Dahlander L, Wallin MW. A man on the inside: unlocking communities as complementary assets. Res Policy. 2006;35(8):1243–59. https://doi.org/10.1016/j.respol.2006.09.011.
3. Watson RT, Boudreau M-C, York PT, Greiner ME, Wynn D Jr. The business of open source. Commun ACM. 2008;51(4):41–6. https://doi.org/10.1145/1330311.1330321.
4. De Noni I, Ganzaroli A, Orsi L. The evolution of Oss governance: a dimensional comparative analysis. Scand J Manag. 2013;29(3):247–63. https://doi.org/10.1016/j.scaman.2012.10.003.
5. West J, O'Mahony S. The role of participation architecture in growing sponsored open source communities. Ind Innov. 2008;15(2):145–68. https://doi.org/10.1080/13662710801970142.
6. Dahlander L, Magnusson M. How do firms make use of open source communities? Long Range Plan. 2008;41(6):629–49. https://doi.org/10.1016/j.lrp.2008.09.003.
7. Colombo MG, Piva E, Rossi-Lamastra C. Open innovation and within-industry diversification in small and medium enterprises: the case of open source software firms. Res Policy. 2014;43(5):891–902. https://doi.org/10.1016/j.respol.2013.08.015.
8. Dahlander L, Magnusson MG. Relationships between open source software companies and communities: observations from nordic firms. Res Policy. 2005;34(4):481–93. https://doi.org/10.1016/j.respol.2005.02.003.
9. Alves C, de Oliveira JAP, Jansen S. Software ecosystems governance-a systematic literature review and research agenda. In: ICEIS 2017-proceedings of the 19th international conference on Enterprise information systems, vol. 3; 2017. p. 26–9. https://doi.org/10.5220/0006269402150226.
10. Franco-Bedoya O, Ameller D, Costal D, Franch X. Open source software ecosystems: a systematic mapping. Inf Softw Technol. 2017;91:160–85. https://doi.org/10.1016/j.infsof.2017.07.007.
11. Teixeira J, Robles G, Gonzlez-Barahona JM. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. J Internet Serv Appl. 2015;6(1) https://doi.org/10.1186/s13174-015-0028-2.
12. West J, Gallagher S. Challenges of open innovation: the paradox of firm investment in open-source software. R D Manag. 2006;36(3):319–31. https://doi.org/10.1111/j.1467-9310.2006.00436.x.
13. Müller-Seitz G, Reger G. Is open source software living up to its promises? Insights for open innovation management from two open source software-inspired projects. R D Manag. 2009;39(4):372–81. https://doi.org/10.1111/j.1467-9310.2009.00565.x.
14. Laursen K, Salter AJ. (2014). The paradox of openness: appropriability, external search and collaboration. Res Policy. 2014;43(5):867–78. https://doi.org/10.1016/j.respol.2013.10.004.
15. Schaarschmidt M, Walsh G, von Kortzfleisch HFO. How do firms influence open source software communities? A framework and empirical analysis of different governance modes. Inf Organ. 2015;25(2):99–114. https://doi.org/10.1016/j.infoandorg.2015.03.001.
16. Felin T, Zenger TR. Closed or open innovation? Problem solving and the governance choice. Res Policy. 2014;43(5):914–25. https://doi.org/10.1016/j.respol.2013.09.006.
17. Gonzalez-Barahona JM, Robles G. Trends in free, libre, open source software communities: from volunteers to companies. It–information technology it–information. Technology. 2013;55(5):173–80. https://doi.org/10.1515/itit.2013.1012.
18. Bosch J. From software product lines to software ecosystems. In: Proceedings of the 13th international software product line conference: Carnegie Mellon University; 2009. p. 111–9.
19. Benlian A, Hilkert D, Hess T. How open is this platform? The meaning and measurement of platform openness from the complementors' perspective. J Inf Technol. 2015;30(3):209–28. https://doi.org/10.1057/jit.2015.6.
20. Baars A, Jansen S. A framework for software ecosystem governance. In: International conference of software business; 2012. p. 168–80. https://doi.org/10.1007/978-3-642-30746-1-14. Springer.
21. Shah SK. Motivation, governance, and the viability of hybrid forms in open source software development. Manag Sci. 2006;52(7):1000–14. https://doi.org/10.1287/mnsc.1060.0553.
22. O'Mahony S. The governance of open source initiatives: what does it mean to be community managed? J Manag Govern. 2007;11(2):139–50. https://doi.org/10.1007/s10997-007-9024-7.
23. Munir H, Wnuk K, Runeson P. Open innovation in software engineering: a systematic mapping study. Empir Softw Eng. 2016;21(2):684–723. https://doi.org/10.1007/s10664-015-9380-x.
24. Mäenpää H, Kilamo T, Mikkonen T, Männistö T. Designing for participation: three models for developer involvement in hybrid Oss projects. In: Open source systems: towards robust practices: Springer International Publishing; 2017. p. 23–33. https://doi.org/10.1007/978-3-319-57735-7-3.
25. Santos CDd. Changes in free and open source software licenses: managerial interventions and variations on project attractiveness. J Int Serv Appl. 2017;8(1):11. https://doi.org/10.1186/s13174-017-0062-3.
26. Parker G, Alstyne MV. Innovation, openness, and platform control. Manag Sci. 2017; https://doi.org/10.1287/mnsc.2017.2757.
27. Markus ML. The governance of free/open source software projects: monolithic, multidimensional, or configurational? J Manag Govern. 2007;11(2):151–63. https://doi.org/10.1007/s10997-007-9021-x.
28. Shaikh M, Henfridsson O. Governing open source software through coordination processes. Inf Organ. 2017;27(2):116–35. https://doi.org/10.1016/j.infoandorg.2017.04.001.
29. Laffan L. A new way of measuring openness: the open governance index. Technol Innov Manag Rev. 2012;2:18–24.
30. Star SL. Distributed artificial intelligence (vol. 2): Morgan Kaufmann Publishers Inc; 1989. p. 37–54. Chap. The Structure of Ill-structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving
31. Steinmacher I, Wiese IS, Conte T, Gerosa MA, Redmiles D. The hard life of open source software project newcomers. In: Proceedings of the 7th international workshop on cooperative and human aspects of software engineering; 2014. p. 72–8. https://doi.org/10.1145/2593702.2593704. ACM.
32. Bettenburg N, Hassan AE, Adams B, German DM. Management of community contributions. Empir Softw Eng. 2015;20(1):252–89. https://doi.org/10.1007/s10664-013-9284-6.
33. Fagerholm F, Guinea AS, Munch J, Borenstein J. The role of mentoring and project characteristics for onboarding in open source software projects. In: Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement; 2014. p. 55. https://doi.org/10.1145/2652524.2652540.
34. Yin RK. Case study research: design and methods, 5th edn. Sage publications. 2014; https://doi.org/10.3138/cjpe.30.1.108.
35. Eisenhardt KM. Building theories from case study research. Acad Manag Rev. 1989;14(4):532–50.
36. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng. 2009;14(2):131–64. https://doi.org/10.1007/s10664-008-9102-8.
37. Gonzalez-Barahona J, Robles G, Izquierdo D, Maffulli S. Using software analytics to understand how companies interact in free software communities. IEEE Softw. 2013;30(5):38–45. https://doi.org/10.1109/MS.2013.95.
38. Lindman J, Hammouda I. Support mechanisms provided by floss foundations and other entities. J Intern Serv Appl. 2018;9:1–12.
39. Kilamo T, Rahikkala J, Mikkonen T. Spicing up open source development with a touch of crowdsourcing. In: 2015 41st Euromicro conference on software engineering and advanced applications; 2015. p. 390–7. https://doi.org/10.1109/SEAA.2015.33.
40. Snow CC, Fjeldstad ØD, Lettl C, Miles RE. Organizing continuous product development and commercialization: the collaborative community of firms model. J Prod Innov Manag. 2011;28(1):3–16.

41. Lamastra CR. Software innovativeness. A comparison between proprietary and free/open source solutions offered by italian smes. R D Manag. 2009; 39(2):153–69.
42. Stuermer M, Spaeth S, Von Krogh G. Extending private-collective innovation: a case study. R D Management. 2009;39(2):170–91.
43. Ritala P, Huizingh E, Almpanopoulou A, Wijbenga P. Tensions in r&d networks: implications for knowledge search and integration. Technol Forecast Soc Chang. 2017;120(Supplement C):311–22. https://doi.org/10.1016/j.techfore.2016.12.020.
44. OpenHub.org public directory of free and Open Source software (FOSS) projects. https://www.openhub.net. Accessed on 23 4 2018.